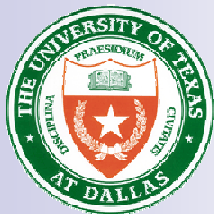


# ***Verilog* HDL – II :** **Sequential Logic**



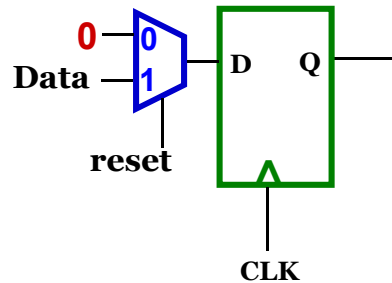
*Poras T. Balsara & Dinesh K. Bhatia*  
Center for Integrated Circuits and Systems  
Department of Electrical Engineering  
University of Texas at Dallas



## D Flip-Flop w/ Asynch Reset in Verilog

```
module dff_async (data, clock, reset, q);  
    input    data, clock, reset;  
    output   q;  
    reg      q;  
  
    always @ (posedge clock or reset)  
    begin  
        if (reset == 1'b0)  
            q <= 1'b0;  
        else  
            q <= data;  
        end  
    endmodule
```

# D Flip-Flop w/ Synch Reset in Verilog



```
module dff_sync (data, clock, reset, q);  
    input    data, clock reset;  
    output   q;  
    reg      q;  
  
    always @ (posedge clock)  
    begin  
        if (reset == 1'b0)  
            q <= 1'b0;  
        else  
            q <= data;  
        end  
    endmodule
```



# Verilog Testbench for D Flip-flop

```
module test_dff (clk);  
    input clk;  
    reg    clk;  
  
    initial  
    begin  
        clk = 0;  
        forever begin  
            #5 clk = ~clk;    // clock period is 10 time units  
        end  
    end  
  
    // Add other statements for setting  
    // data and reset inputs  
  
    ...    ...  
  
endmodule
```



# Sequential Logic Design Using Verilog

- **Example:** Use Verilog HDL to design a sequence detector with one input  $X$  and one output  $Z$ . The detector should recognize the input sequence “101”. The detector should keep checking for the appropriate sequence and should not reset to the initial state after it has recognized the sequence. The detector initializes to a reset state when input, *RESET* is activated.

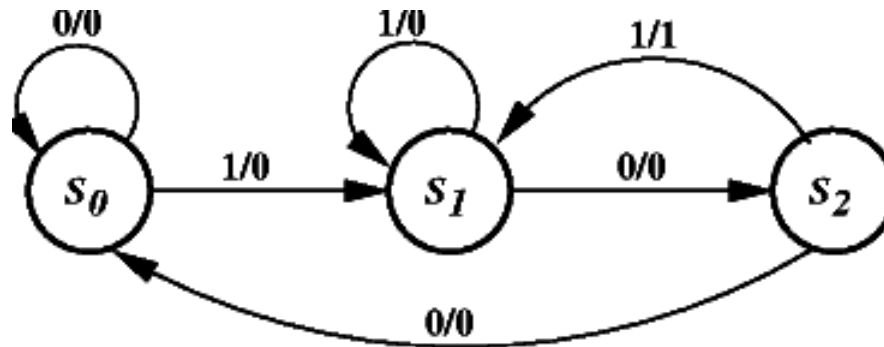
*For example, for input*

$X = \text{“...110110101...”}$ ,

*Mealy output*

$Z = \text{“...000100101...”}$

## Example: *Mealy* Machine Implementation



Present State $Q_1 Q_0$	Next State $Q_1^* Q_0^*$		Output ( $z$ )	
	$x = 1$	$x = 0$	$x = 1$	$x = 0$
00	01	00	0	0
01	01	10	0	0
10	01	00	1	0

# Mealy Machine in Verilog HDL

```
`define CK2Q 5 // Defines the Clock-to-Q Delay of the flip flop.

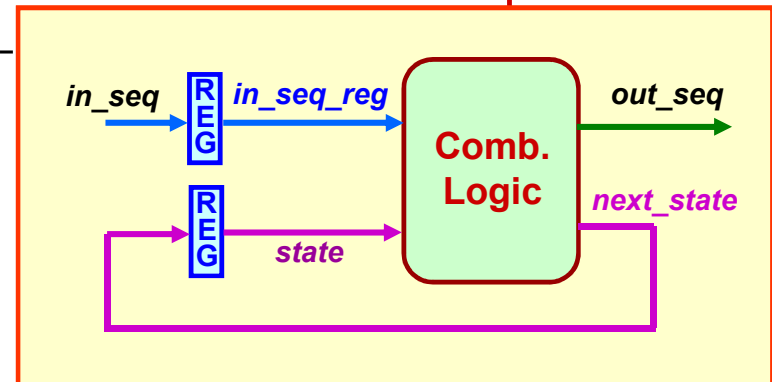
module mealy_fsm(reset,clk,in_seq,out_seq);
  input reset;
  input clk;
  input in_seq;
  output out_seq;

  reg out_seq;
  reg in_seq_reg;

  //----- Parameters defining State machine States-----
  parameter SIZE = 2;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;

  //----- Internal Variables -----
  reg [SIZE-1:0] state; // Memory part of the FSM
  reg [SIZE-1:0] next_state; // Combinational part of the FSM

  //-----Register the input -----
  always @ (posedge clk)
  begin : REG_INPUT
    if (reset == 1'b1)
      in_seq_reg <= #`CK2Q 1'b0;
    else
      in_seq_reg <= #`CK2Q in_seq;
  end
```

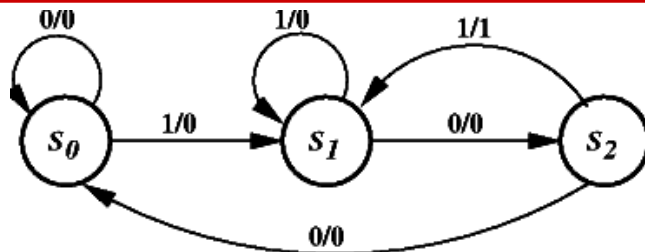


# Mealy Machine in Verilog HDL (contd...)

```

always @ (state or in_seq_reg or reset)
begin : FSM_COMBO
    next_state = 2'b00;
    if (reset == 1'b1)
        out_seq <= 1'b0;
    else begin
        case(state)
            S0 : if (in_seq_reg == 1'b0) begin
                    next_state = S0;
                    out_seq <= 1'b0;
                end
                else begin
                    next_state = S1;
                    out_seq <= 1'b0;
                end
            S1 : if (in_seq_reg == 1'b0) begin
                    next_state = S2;
                    out_seq <= 1'b0;
                end
                else begin
                    next_state = S1;
                    out_seq <= 1'b0;
                end
        endcase
    end
end

```



```

        S2 : if (in_seq_reg == 1'b0) begin
                next_state = S0;
                out_seq <= 1'b0;
            end
            else begin
                next_state = S1;
                out_seq <= 1'b1;
            end
        default : begin
                next_state = S0;
                out_seq <= 1'b0;
            end
        endcase
    end

    // Register combinational "next_state" variable

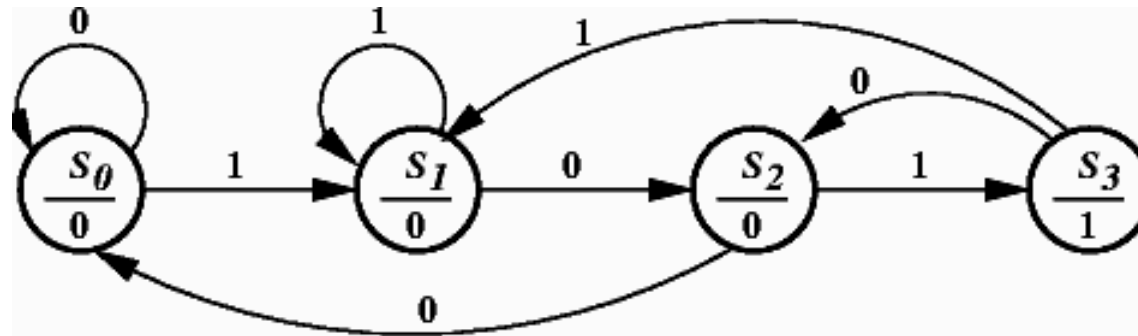
    always @ (posedge clk)
    begin : FSM_SEQ
        if (reset == 1'b1)
            state <= #`CK2Q S0;
        else
            state <= #`CK2Q next_state;
        end
    end

endmodule

```



## Example: Moore Machine Implementation



```
`define CK2Q 5 // Defines the Clock-to-Q Delay of the flip-flop
module moore_fsm(reset,clk,in_seq,out_seq);
    input reset;
    input clk;
    input in_seq;
    output out_seq;
    reg out_seq;

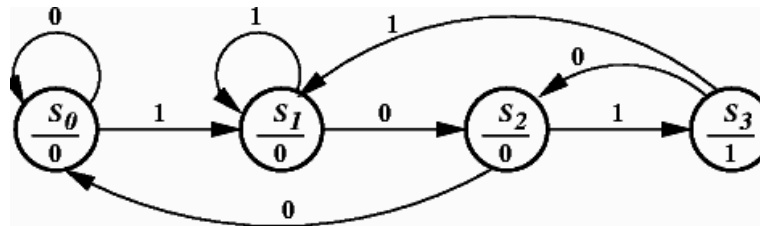
    //----- Parameters defining State machine States -----
    parameter SIZE = 4;
    parameter S0 = 4'b0001, S1 = 4'b0010, S2 = 4'b0100, S3 = 4'b1000;

    //----- Internal Variables -----
    reg [SIZE-1:0] state; // Sequential part of the FSM
    reg [SIZE-1:0] next_state; // Combinational part of the FSM
```

# Moore Machine in Verilog HDL (contd...)

```
always @ (state or in_seq)
begin : FSM_COMBO
  next_state = 4'b0001;
  case(state)
    S0 : if (in_seq == 1'b1)
          next_state = S1;
        else
          next_state = S0;
    S1 : if (in_seq == 1'b0)
          next_state = S2;
        else
          next_state = S1;
    S2 : if (in_seq == 1'b1)
          next_state = S3;
        else
          next_state = S0;
    S3 : if (in_seq == 1'b1)
          next_state = S1;
        else
          next_state = S2;
    default : next_state = S0;
  endcase
end
```

```
always @ (posedge clk)
begin : FSM_SEQ
  if (reset == 1'b1)
    state <= #`CK2Q S0;
  else
    state <= #`CK2Q next_state;
  end
always @ (state or reset)
begin : OUTPUT_LOGIC
  if (reset == 1'b1)
    out_seq <= 1'b0;
  else begin
    case(state)
      S0 : out_seq <= 1'b0;
      S1 : out_seq <= 1'b0;
      S2 : out_seq <= 1'b0;
      S3 : out_seq <= 1'b1;
      default: out_seq <= 1'b0;
    endcase
  end
end // End Of Block OUTPUT_LOGIC
endmodule // End of Moore state machine
```



# Verilog Testbenches for Sequential Circuits

```
module test_fsm (reset,clk,in_seq,out_seq);
    output reset, clk, in_seq;
    reg    reset, clk, in_seq;
    input  out_seq;

    reg [15:0] data;
    integer i;

    // The input data sequence is defined in the bit
    // vector "data". On each clock one bit of data is
    // sent to the state machine which will detect the
    // sequence "101" in this data

    initial          // Initialization
    begin
        data = 16'b0010100110101011;
        i = 0;
        reset = 1'b1;    // reset the state machine
        #1200;
        reset = 1'b0;
        #20000;
        $finish;
    end
```

```
initial          // Clock generation
begin
    clk = 0;
    forever begin
        #600;
        clk = ~clk;
    end
end

// Right shifting of data to
// generate the input sequence

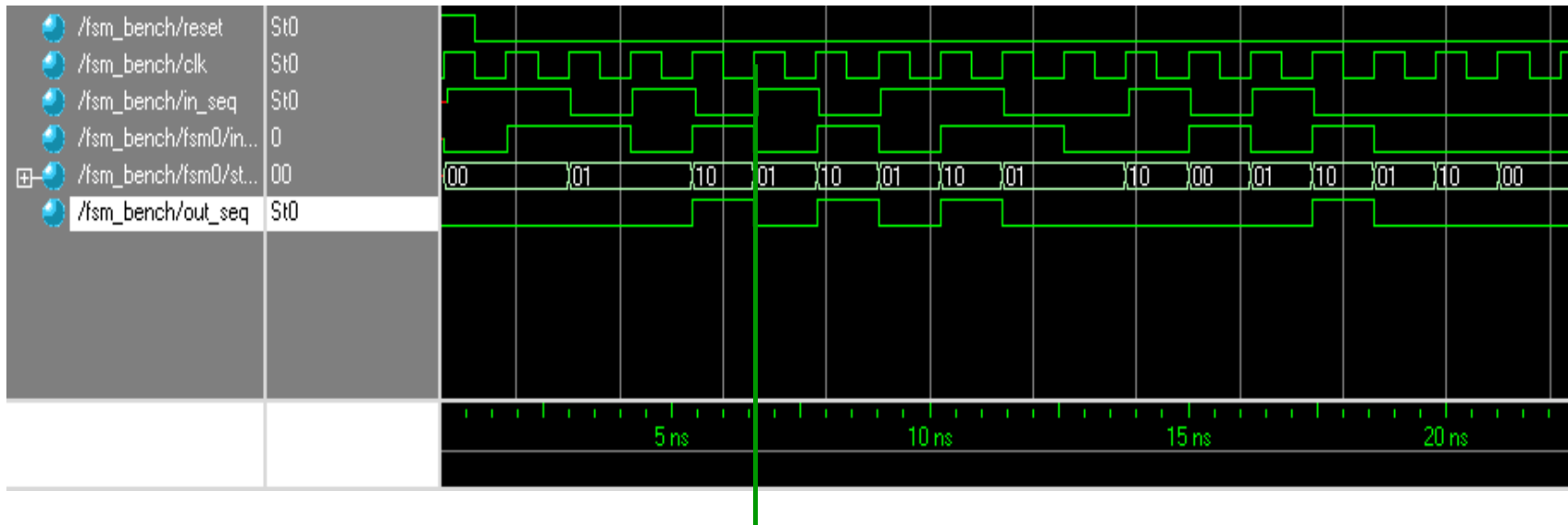
always @ (posedge clk)
begin
    #50;
    in_seq = data >> i;
    i = i+1;
end

endmodule
```

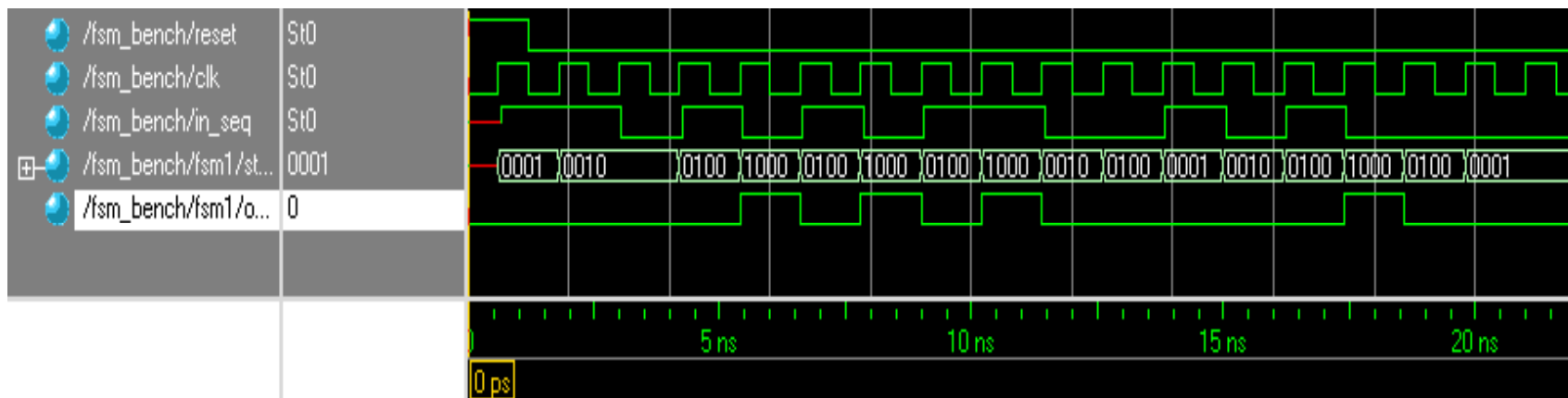
```
module fsm_bench ();
    wire clk, reset, in_seq, out_seq;

    test_fsm test (reset,clk,in_seq,out_seq);
    mealy_fsm fsm0 (reset,clk,in_seq,out_seq);
endmodule
```

# Mealy Machine Simulation Results



# Moore Machine Simulation Results

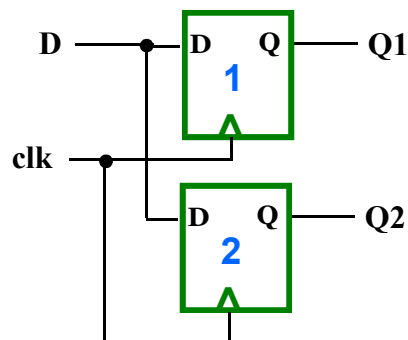


# Blocking (=) vs Non-Blocking (<=)

## ■ Blocking Statement:

```
module block (D, clk, Q1, Q2)
  input  D, clk;
  output Q1, Q2;
  reg    Q1, Q2;

  always @ (posedge clk)
  begin
    Q1 = D;
    Q2 = Q1;
  end
endmodule
```



## ■ Non-Blocking Statement:

```
module block (D, clk, Q1, Q2)
  input  D, clk;
  output Q1, Q2;
  reg    Q1, Q2;

  always @ (posedge clk)
  begin
    Q1 <= D;
    Q2 <= Q1;
  end
endmodule
```

